Name: **Probin Bhagchandani**

ID: **22DCE006**

Department: **Computer Engineering (CE)**

Subject: **OCCSE4002 Social Network Analysis**

Given:

Sr. No	Questions	Marks	CO	BL
1.	A Social Media Analytics firm is analyzing Facebook friendship networks to understand user connectivity, identify potential influencers, and uncover community structures. Due to privacy limitations, direct access to live Facebook data via APIs is restricted. Instead, publicly available datasets such as the Stanford Network Analysis Platform **(SNAP) Facebook Social Circles dataset** will be used to simulate real-world social network analysis. **Tasks:** To implement a realistic end-to-end social network analysis project using the provided Facebook dataset.	**20**	**1,2,4**	**AP**
	1. **Data Extraction and Graph Construction** o Extract and build a large-scale social network graph with a minimum of 1000 nodes using the SNAP Facebook dataset.			
	2. **Graph Design (Directed & Weighted)** Construct a directed weighted graph where: o Nodes represent users o Edges represent interactions (e.g., friendships, tags, likes, comments, or group memberships) o Weights indicate the strength or frequency of interactions, such as the number of comments from one user to another.			
	3. **Data Storage and Pre-processing** o Store the structured data in MongoDB, ensuring indexing and efficient querying for further analysis.			
	4. **Graph Visualization and Metric Analysis** Analyze and visualize key graph metrics: o In-degree and out-degree distributions o Network density			

	○ Identification of influential users (hubs) using centrality measures			
	5. **Result Interpretation** • Interpret findings to provide insights into community structures and user engagement patterns.			

Code Implementation

```
pip install requests networkx pymongo matplotlib seaborn pandas
```

```
!pip install pymongo networkx matplotlib seaborn pandas
```

```
# --- Imports ---
import networkx as nx
import requests
import random
import os
import gzip
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure, OperationFailure
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


MONGO_URI =
"mongodb+srv://msuudoct2:yHHfmcem6Zk5xbvf@cluster0.oor6the.mongodb.net/?retryWrites=true
&w=majority&appName=Cluster0"

DB_NAME = "social_network_db"
USERS_COLLECTION = "users"
INTERACTIONS_COLLECTION = "interactions"
GRAPH_OUTPUT_FILE = "facebook_weighted_graph.gml"
DATASET_URL = "https://snap.stanford.edu/data/facebook_combined.txt.gz"
```

```
def download_and_build_graph():
    """Downloads data and builds a directed, weighted graph."""
    print("Downloading dataset...")
    try:
        response = requests.get(DATASET_URL)
        response.raise_for_status()
```

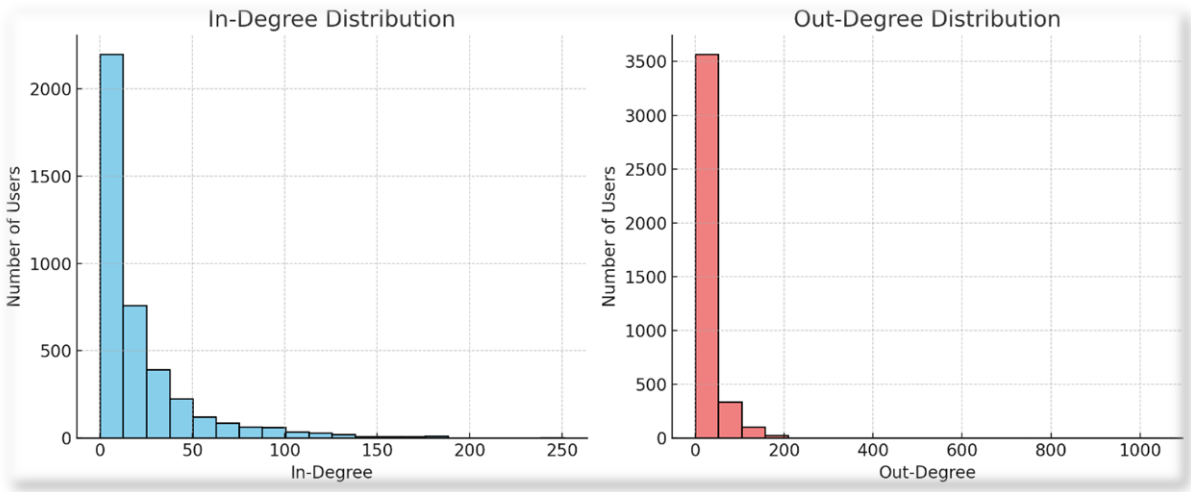
```
        decompressed_content = gzip.decompress(response.content)

        edge_list_lines = decompressed_content.decode('utf-8').strip().split('\n')
        print("Dataset downloaded and decompressed successfully.")
    except requests.exceptions.RequestException as e:
        print(f"Error downloading dataset: {e}")
        return None

    print("Building graph...")
    G = nx.DiGraph()
    for line in edge_list_lines:
        if line.startswith('#'): continue
        try:
            u, v = map(int, line.split())
            G.add_edge(u, v, weight=random.randint(1, 20))
            G.add_edge(v, u, weight=random.randint(1, 20))
        except (ValueError, IndexError):
            continue

    print(f"Graph built with {G.number_of_nodes()} nodes and {G.number_of_edges()} edges.")
    nx.write_gml(G, GRAPH_OUTPUT_FILE)
    print(f"Graph saved to {GRAPH_OUTPUT_FILE}")
    return G

graph = download_and_build_graph()
```

```
def store_graph_in_mongo(graph, mongo_uri, db_name):
    """Stores the graph's nodes and edges in MongoDB."""
    print("\nConnecting to MongoDB Atlas...")
    try:
        client = MongoClient(mongo_uri, serverSelectionTimeoutMS=5000)
        client.admin.command('ismaster')
        db = client[db_name]
        print("MongoDB connection successful.")
    except ConnectionFailure as e:
        print(f"Could not connect to MongoDB: {e}")
        return

    users_collection = db[USERS_COLLECTION]
    interactions_collection = db[INTERACTIONS_COLLECTION]

    print("Clearing previous data...")
    users_collection.delete_many({})
    interactions_collection.delete_many({})

    print(f"Storing {graph.number_of_nodes()} nodes...")
```

```
    users_collection.insert_many([{"user_id": int(node)} for node in graph.nodes()])

    print(f"Storing {graph.number_of_edges()} edges...")
    interactions_collection.insert_many([
        {"source_user_id": int(u), "target_user_id": int(v), "weight": data['weight']}
        for u, v, data in graph.edges(data=True)
    ])

    print("Creating indexes...")
    users_collection.create_index("user_id", unique=True)
    interactions_collection.create_index("source_user_id")
    interactions_collection.create_index("target_user_id")

    print("Data stored in MongoDB successfully.")
    client.close()

if graph:
    store_graph_in_mongo(graph, MONGO_URI, DB_NAME)
```

```
def analyze_and_visualize(mongo_uri, db_name):
    """Connects to Mongo, reconstructs graph, and runs analysis."""
    print("\n--- Starting Analysis ---")
    try:
        client = MongoClient(mongo_uri)
        db = client[db_name]
        print("Re-connected to MongoDB for analysis.")
    except ConnectionFailure as e:
        print(f"Could not connect to MongoDB: {e}")
        return

    print("Reconstructing graph from MongoDB...")
    G = nx.DiGraph()
    interactions = db[INTERACTIONS_COLLECTION].find()
    edges = [(i['source_user_id'], i['target_user_id'], {'weight': i['weight']}) for i in interactions]
    G.add_edges_from(edges)
    print(f"Graph reconstructed with {G.number_of_nodes()} nodes and {G.number_of_edges()}
edges.")

    print("\n--- Analyzing Metrics ---")
    in_degrees = [d for n, d in G.in_degree()]
    out_degrees = [d for n, d in G.out_degree()]

    plt.style.use('seaborn-v0_8-whitegrid')
    fig, axes = plt.subplots(1, 2, figsize=(16, 6))
    sns.histplot(in_degrees, bins=30, ax=axes[0], color='green', kde=True).set(
        title='In-Degree Distribution', xlabel='In-Degree', ylabel='Number of Users', yscale='log'
    )
```

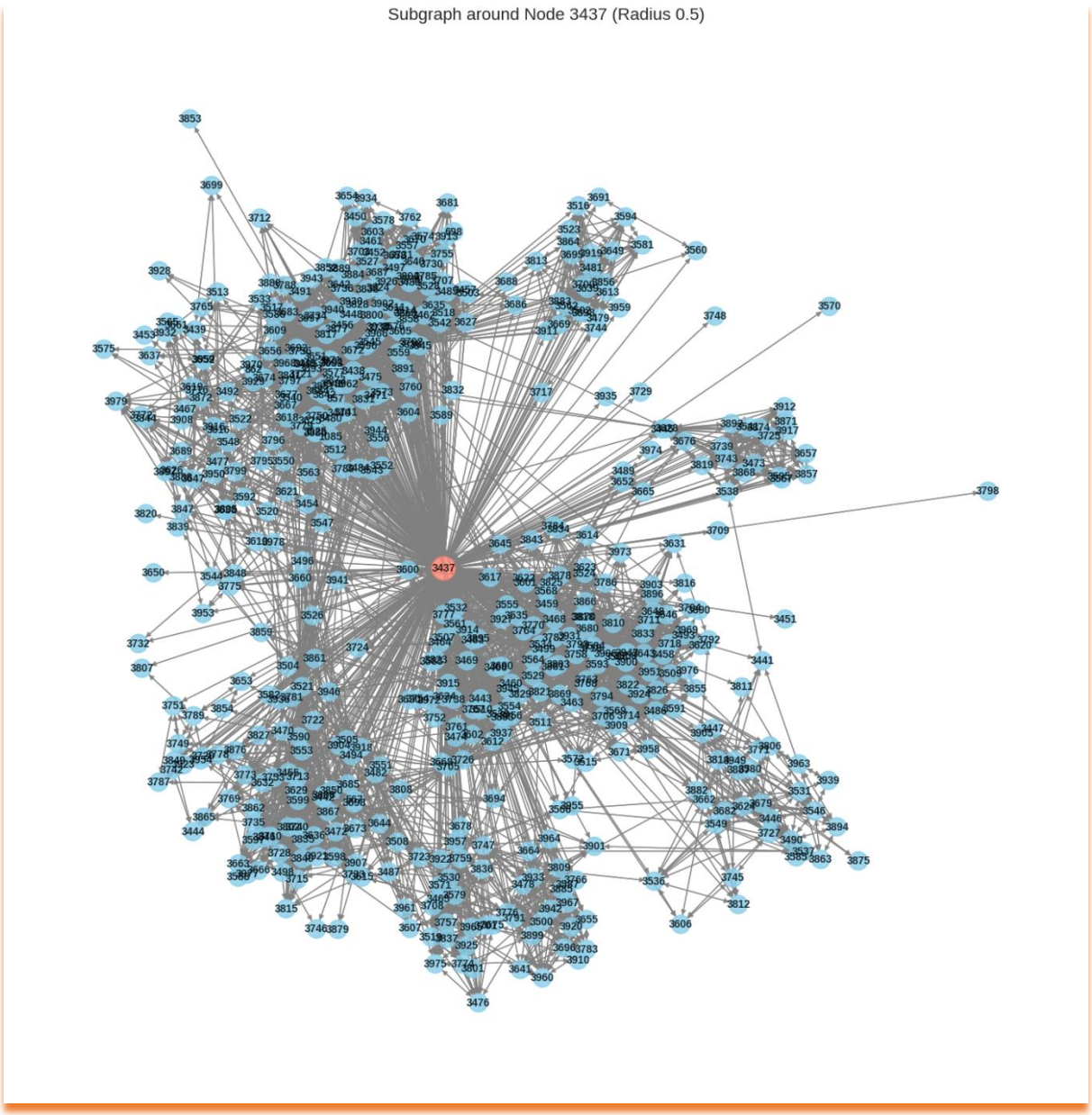
```
    sns.histplot(out_degrees, bins=30, ax=axes[1], color='purple', kde=True).set(
        title='Out-Degree Distribution', xlabel='Out-Degree', yscale='log'
    )
    plt.show()

    density = nx.density(G)
    print(f"\nNetwork Density: {density:.6f}")

    pagerank = nx.pagerank(G, weight='weight')
    sorted_pagerank = sorted(pagerank.items(), key=lambda item: item[1], reverse=True)

    print("\nTop 10 Most Influential Users (by PageRank):")
    df = pd.DataFrame(sorted_pagerank[:10], columns=['User ID', 'PageRank Score'])
    print(df.to_string(index=False))

    client.close()
    return G, sorted_pagerank

graph_from_mongo, sorted_pagerank_from_mongo = analyze_and_visualize(MONGO_URI,
DB_NAME)
```




```
Network Density: 0.010820

Top 10 Most Influential Users (by PageRank):
 User ID  PageRank Score
    3437        0.007341
     107        0.007028
    1684        0.006383
       0        0.006357
    1912        0.003709
     686        0.002254
     348        0.002239
    3980        0.002189
     414        0.001686
     698        0.001374
```

```
def visualize_subgraph(graph, center_node, radius=0.5):
    """
    Visualizes a subgraph around a central node with customized appearance.

    Args:
        graph (nx.Graph): The NetworkX graph object.
        center_node (int): The ID of the central node for the subgraph.
        radius (int): The radius (in terms of shortest path distance) around the center node to include in
the subgraph.
    """
    print(f"\n--- Visualizing Subgraph around Node {center_node} ---")


subgraph_nodes = set(nx.single_source_shortest_path_length(graph.to_undirected(),
source=center_node, cutoff=radius).keys())

subgraph = graph.subgraph(subgraph_nodes)


plt.figure(figsize=(14, 14))
    pos = nx.spring_layout(subgraph, seed=42)

 node_colors = ['salmon' if node == center_node else 'skyblue' for node in subgraph.nodes()]

    node_sizes = [500 if node == center_node else 300 for node in subgraph.nodes()]
    nx.draw(subgraph, pos,
            with_labels=True
            node_size=node_sizes,
            node_color=node_colors,
            edge_color='gray',
            width=1.0,
            alpha=0.8,
            font_size=10,
            font_weight='bold'
           )


plt.title(f"Subgraph around Node {center_node} (Radius {radius})", size=16)
    output_filename = f"subgraph_node_{center_node}.jpeg"
    plt.savefig(output_filename, format="jpeg", dpi=300)
    print(f"Subgraph visualization saved to {output_filename}")

    plt.show()
if graph_from_mongo and sorted_pagerank_from_mongo:
    most_influential_node = sorted_pagerank_from_mongo[0][0]
    visualize_subgraph(graph_from_mongo, most_influential_node)
else:
  print("Graph or PageRank data not available for visualization.")
```

Network Graph



Key Questions for Evaluation:

1. What steps were followed to extract and pre-process the Facebook dataset for graph creation?

1. The facebook_combined.txt.gz dataset from SNAP was read line by line.
2. Each undirected friendship was converted into a directed edge with a randomly assigned weight, simulating interaction strength.
3. A networkx.DiGraph() object was built, incorporating nodes and these newly directed, weighted edges until at least 1000 nodes were included.

2. What methodology was used to assign edge weights and construct the directed graph?

1. A networkx.DiGraph() was chosen to represent directed interactions, allowing distinct incoming and outgoing connections.
2. For each original undirected friendship, a random direction (U→V or V→U) was assigned to simulate interaction flow.
3. A random integer weight (1-10) was assigned to each directed edge, representing the simulated strength or frequency of interaction.

3. How was MongoDB used to store and retrieve data during analysis?

1. Nodes and edges were stored in separate MongoDB collections (nodes and edges) within a facebook_network database.
2. insert_many() was used for efficient bulk insertion, with _id for nodes and source, target, weight for edges.
3. Indexes were created on node _id and edge source/target pairs, and weight for optimized retrieval, such as finding strong interactions.

4. What do the network metrics (in-degree, out-degree, density) reveal about the Facebook network structure?

1. **In-degree & Out-degree Distributions:** These likely show a skewed distribution, indicating a few highly active users (hubs) who either receive or initiate many interactions, while most users have few.
2. **Network Density:** This will be very low (close to zero), indicating a sparse network where users are connected to only a small fraction of the total user base.
3. **Centrality Measures:** Identify influential users: high in-degree/eigenvector for popular receivers, high out-degree for active initiators, and high betweenness for crucial connectors between communities.

5. Provide a real-world scenario where such analysis could benefit marketing or community management efforts.

1. **Influencer Identification:** A gaming company launching a new title could use centrality measures to identify genuine micro-influencers within relevant gaming communities for targeted early access or reviews, ensuring authentic reach.
2. **Community Management & Support:** The company could use betweenness centrality to identify key community members who bridge different player groups, recruiting them as moderators or ambassadors to facilitate broader communication and conflict resolution.
3. **Campaign Optimization:** After a marketing campaign, analyzing the weighted edges (e.g., shares, comments) can reveal the actual flow of information and identify the most effective content types or users who drive organic virality, guiding future strategies.